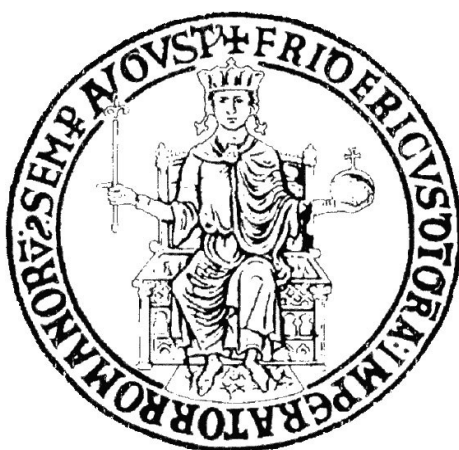


Gara di corsa in un campo minato



Alberto Panzera
Chiara Pellecchia

Indice

1. Descrizione del progetto

2. Guida d'uso

- Compilazione ed esecuzione
- Gameplay
- Comunicazione client server nelle fasi di gioco

3. Dettagli implementativi e strategie risolutive

4. Codice sorgente

1. Descrizione del progetto

Descrizione sintetica del problema:

Realizzare un sistema client-server che consenta a più utenti di giocare ad una gara di corsa in un campo minato. Si utilizzi il linguaggio C su piattaforma UNIX. I processi dovranno comunicare tramite socket TCP.

L'ambiente di gioco è una matrice che contiene informazioni riguardanti ostacoli e giocatori, posizionati dal server in modo casuale. I primi sono nascosti e verranno visualizzati dal client alla sola collisione, i secondi sono visibili in ogni momento nelle loro posizioni aggiornate.

Ogni giocatore è rappresentato da una coppia di coordinate sulla mappa ed ha a disposizione una lista di azioni che comprende:

- spostamento in tutte le direzioni (N,S,E,O),
- visualizzazione del tempo rimanente,
- visualizzazione della lista di utenti online e delle loro coordinate,
- possibilità di logout.

Alla collisione con una mina, il giocatore sarà eliminato dal gioco lasciando agli altri la possibilità di vincere. La vittoria è possibile solo al raggiungimento del traguardo, ovvero una posizione qualunque dell'ultima colonna.

Alla scadenza di un tempo prefissato, la sessione di gioco terminerà e tutti gli utenti saranno eliminati e potranno partecipare ad una prossima sessione dopo che il server avrà generato una nuova mappa in una nuova partita.

2. Guida d'uso

Compilazione ed esecuzione

Attraverso il compilatore **gcc** è possibile compilare il file `Server.c` in questo modo:

```
$ gcc server.c -lpthread -o server
```

ed eseguirlo con `./server` seguito da *numero di porta* da command line:

```
$ ./server 2000
```

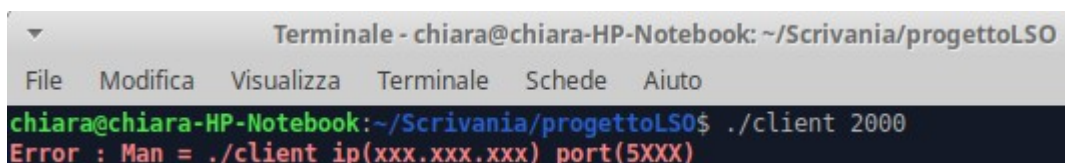
La stessa cosa avviene per il client:

```
$ gcc client.c -g -o client  
$
```

esso viene eseguito con *indirizzo ip* e *numero di porta*:

```
$ ./client localhost 2000
```

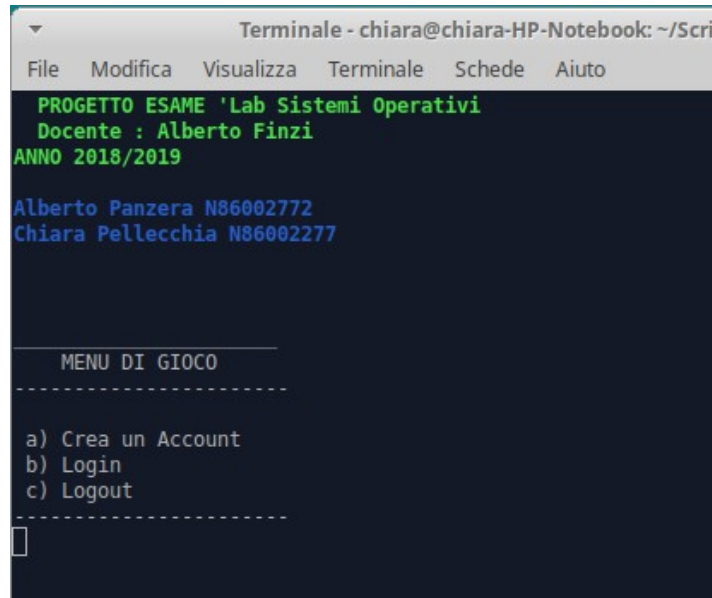
È stato gestito il caso di errore all'inserimento di:



The screenshot shows a terminal window titled "Terminale - chiara@chiara-HP-Notebook: ~/Scrivania/progettoLSO". The menu bar includes "File", "Modifica", "Visualizza", "Terminale", "Schede", and "Aiuto". The prompt is "chiara@chiara-HP-Notebook:~/Scrivania/progettoLSO\$". The user has entered the command `./client 2000`. The terminal displays an error message in red text: `Error : Man = ./client ip(xxx.xxx.xxx) port(5XXX)`.

Menù principale

Una volta eseguito il file client.c l'utente sarà indirizzato alla pagina del menù principale, dove è possibile eseguire le seguenti azioni: registrazione, login, logout.



```
Terminale - chiara@chiara-HP-Notebook: ~/Scrivania
File Modifica Visualizza Terminale Schede Aiuto

PROGETTO ESAME 'Lab Sistemi Operativi
Docente : Alberto Finzi
ANNO 2018/2019

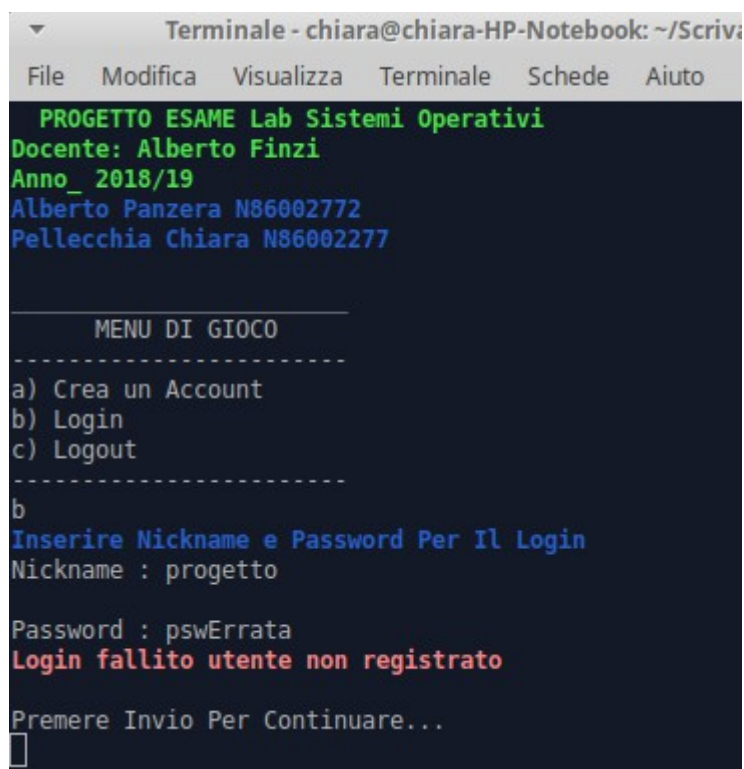
Alberto Panzera N86002772
Chiara Pellecchia N86002277

-----
MENU DI GIOCO
-----

a) Crea un Account
b) Login
c) Logout
-----


```

È stato gestito il caso di fallimento in fase di login, per password o nome utente errati.



```
Terminale - chiara@chiara-HP-Notebook: ~/Scrivania
File Modifica Visualizza Terminale Schede Aiuto

PROGETTO ESAME Lab Sistemi Operativi
Docente: Alberto Finzi
Anno_ 2018/19
Alberto Panzera N86002772
Pellecchia Chiara N86002277

-----
MENU DI GIOCO
-----

a) Crea un Account
b) Login
c) Logout
-----

b
Inserire Nickname e Password Per Il Login
Nickname : progetto

Password : pswErrata
Login fallito utente non registrato

Premere Invio Per Continuare...

```

Gameplay

Una volta completato il login il gioco mostrerà la schermata di gameplay, in cui è compresa la visualizzazione della mappa nella partita in cui è stato inserito (comune a tutti i client). Nella mappa 20x20 c'è un numero di bombe pari a 8, queste sono nascoste mentre è possibile visualizzare gli altri utenti e i loro movimenti.

All'inizio del gioco l'utente sarà posizionato in una posizione randomica sulla prima colonna. In ogni client, il proprio personaggio è rappresentato da una '@' blu mentre tutti gli avversari sono rappresentati da '@' bianche.

```
      0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
0[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
1[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
2[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
3[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
4[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
5[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
6[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
7[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
8[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
9[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
10[ @  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
11[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
12[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
13[ -  -  -  @  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
14[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
15[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
16[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
17[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
18[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
19[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  ]
w) Sopra a) Sinistra s) Sotto d) Destra t) Tempo i) Info c) Logout
Posizione [13][3] ID partita: [1])
█
```

I comandi possibili sono:

- **w** spostamento verso l'alto
- **a** spostamento a sinistra
- **s** spostamento in basso
- **d** spostamento a destra
- **i** informazioni riguardo agli utenti online e le loro coordinate
- **t** tempo trascorso e rimanente

In corrispondenza del comando t:

```
Ti trovi in Posizione [3][8] (Stai partecipando alla partita ID: [1])
t
Tempo :11 secondi

Mancano ancora :109 secondi

Premere Invio Per Continuare...
█
```

In corrispondenza del comando i:

```
      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
0[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
1[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
2[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
3[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
4[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
5[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
6[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
7[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
8[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
9[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
10[ @  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
11[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
12[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
13[ -  -  -  @  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
14[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
15[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
16[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
17[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
18[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
19[ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - ]
w) Sopra  a) Sinistra  s) Sotto  d) Destra t) Tempo  i) Info c) Logout
Posizione [13][3] ID partita: [1])
i
Utenti online:
user  b in posizione  [13][3]

user  a in posizione  [10][0]

Premere Invio Per Continuare...
█
```

Comunicazione client-server

Menù:

La prima scelta a cui è sottoposto l'utente è nel **menù** (a registrazione, b login, c exit). Il client legge il carattere che determina la scelta (vediamo un esempio nel caso del login), e manda una **socket** con nickname e password al Server in questo modo:

```
else if(scelta=='b'){
    pulisciBuffer();

    stampa(BLU"Inserire Nickname e Password Per Il Login\n"RESET);
    stampa("Nickname : ");

    if((bytesnickname=read(1,Nickname,256))<0){
        error("Errore write\n");
    }
    Nickname[bytesnickname-1]='\0';

    stampa("\nPassword : ");
    if((bytepassword=read(1>Password,256))<0){
        error("Errore write\n");
    }
    Password[bytepassword-1]='\0';

    sprintf(Login,"b%s:%s\n",Nickname>Password);
    if(write(sock>Login,strlen(Login))<0){
        error("Errore WRITE\n");
    }
    if(read(sock,&risposta,1)<0){
        error("Error Read\n");
    }
    switch(risposta){
        case 'y':system("clear"); stampa(GREEN"Adesso puoi iniziare a giocare!\n"RESET);
            inGioco=1;
            gioco(sock);
            break;

        case 'n': stampa(RED"Login fallito utente non registrato\n"RESET); pauseT(); system("clear");
            break;

        case 'g': stampa(RED"Login fallito utente già Online\n"RESET); pauseT(); system("clear");
            break;
    }
}
```

client.c

Successivamente, legge la risposta del server (nello switch). La risposta è una 'y' nel caso in cui la comunicazione abbia avuto successo, è una 'n' in caso contrario. Inoltre, viene controllato se l'utente sia già online, in caso affermativo viene inviata una 'g'.

Il server legge la socket con le informazioni riguardanti l'utente e manda la risposta al client.

```
else if(scelta[0]=='b'){
    scelta[nickname]='\0';
    getNickname(Addnickname,scelta+1);
    pthread_mutex_lock(&mymutex);//blocca il mutex
    Online=giaOnline(Addnickname,Ut);
    struct Utente **tmp=&Ut;
    pthread_mutex_unlock(&mymutex);//sblocca il mutex

    if(Online==1){
        dataCorrente(data);
        sprintf(messaggio,"Impossibile effettuare il login l'Utente è gia loggato con %s in data : %s\n",scelta+1,data);
        stampaLog(fdlog_loc,messaggio);
        sendSocket(conn,"g",1);
        Online=0;
    }
    else{
        dataCorrente(data);
        sprintf(messaggio,"L'utente %s si e' connesso in data : %s\n",Addnickname,data);
        stampaLog(fdlog_loc,messaggio);
        if(!checkLogin(scelta+1,fdUtente_loc)){
            sendSocket(conn,"y",1);
            pthread_mutex_lock(&mymutex);//blocca il mutex
            aggiungiUtente(tmp,Addnickname,pthread_self());
            pthread_mutex_unlock(&mymutex);//sblocca il mutex
            if(MappaCreatà==0){
                if(pthread_create(&threadServer,NULL,creaMappa,NULL) < 0){
                    error("Errore Thread\n");
                }
            }
            gioco(conn,Addnickname);
        }else{
            sendSocket(conn,"n",1);
        }
    }
}
```

server.c

In fase di gioco:

Per gestire la collisione con una bomba (X) o con un altro utente (U) il server inserisce un segnale nella posizione della mappa in cui l'utente farà il movimento.

```
if(nuovaPosizione[2]==19){
    vittoria=1;
}
else{
    if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]+1]=='X'){
        coordinateClient[0]=nuovaPosizione[1];
        coordinateClient[1]=nuovaPosizione[2];
        coordinateClient[2]='X';
        pthread_mutex_lock(&mymutex);//blocca il mutex
        Ut=eliminaUtente(Ut,nickname);
        pthread_mutex_unlock(&mymutex);//sblocca il mutex
        flag=0;
    }
    else if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]+1]=='U'){
        coordinateClient[0]=nuovaPosizione[1];
        coordinateClient[1]=nuovaPosizione[2];
        coordinateClient[2]='U';
    }

    else{
        pthread_mutex_lock(&mymutex);//blocco il mutex
        if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=='U'){
            Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=' ';
        }
        coordinateClient[0]=nuovaPosizione[1];
        coordinateClient[1]=nuovaPosizione[2]+1;
        coordinateClient[2]=' ';
        Mappa[nuovaPosizione[1]][nuovaPosizione[2]+1]='U';
        pthread_mutex_unlock(&mymutex);//sblocco il mutex
    }
}
```

server.c

Per gestire la vittoria, invece, il server manda una 'v' tramite socket. Lo stesso metodo è stato usato, attraverso una 'f', per gestire i casi in cui il tempo è scaduto o i client si trovano in partite diverse.

```
if(vittoria==1){
    sendSocket(conn,"v",1);
    pthread_mutex_lock(&mymutex);
    deallocaUtente(Ut);
    Ut=NULL;
    flag=0;
    pthread_mutex_unlock(&mymutex);
}else if(nonvittoria==1){
    sendSocket(conn,"f",1);
    flag=0;
}
```

A seguire, il codice del client in risposta al segnale ricevuto.

```
if(nuovaPosizione[0]=='v'){
    stampaVittoria();
    sleep(1);
    inGioco=0;
    vittoria=1;
    giocato=1;
}else if(nuovaPosizione[0]=='f'){
    system("clear");
    inGioco=0;
    giocato=1;
}

else if(nuovaPosizione[2]=='X'){
    stampaSconfitta();
    sleep(1);
    inGioco=0;
    giocato=0;
}

else{
    Mappa[i][j]='-';
    i=nuovaPosizione[0]; j=nuovaPosizione[1];
    Mappa[i][j]='U';
}
```

client.c

3. Dettagli implementativi

Visualizzazione di tutti gli utenti online:

All'interno della funzione principale del client è presente un costrutto switch-case, in cui i casi corrispondono ad ognuna delle scelte che l'utente può compiere.

In ognuno dei casi riguardanti il movimento, tramite il seguente frammento di codice il client manda un segnale (codificato con 'k') al server.

```
for(k=0;k<5000;k++){
    lista[k]='\0';
}

sendSocket(sock,"k",1);
if(read(sock,lista,5000)<0){
    error("errore read\n");
}
stampaMappa(Mappa,i,j,id_corrente,lista);
```

client.c

Alla ricezione del segnale, il server implementa un caso 'k', in cui viene composta la **stringa** di coordinate da mandare tramite **socket** al client.

La stringa è codificata nel seguente formato (le coordinate del singolo client sono divise da un '?' e tra un client e l'altro intercorre un '&') :

$$\mathbf{x}_1 ? \mathbf{y}_1 \& \mathbf{x}_2 ? \mathbf{y}_2 \& \mathbf{x}_3 ? \mathbf{y}_3$$

```

case 'k':
    pthread_mutex_lock(&mymutex); // blocca il mutex
    struct Utente *tmp2=Ut;
    pthread_mutex_unlock(&mymutex); // sblocca il mutex
    char lista2[5000];
    for(k=0; k<5000; k++){
        lista2[k]='\0';
    }
    while(tmp2!=NULL){
        char coordinateLoc[50];
        sprintf(coordinateLoc, "%d?%d", tmp2->x, tmp2->y);
        strcat(lista2, coordinateLoc);
        strcat(lista2, "&");
        tmp2=tmp2->next;
    }
    lista2[strlen(lista2)-1]='\0';
    sendSocket(conn, lista2, strlen(lista2));
    break;

default :
    flag=0;
    break;

```

client.c

La fase finale, nel server, consiste nel leggere la stringa e decodificarla ricavandone le coordinate dei singoli client.

```

if(otherplayers != NULL){
    // scorro la stringa
    for(in=0; in<strlen(otherplayers); in++){
        // se è un numero
        if(otherplayers[in]!='?' && otherplayers[in]!='&' && otherplayers[in]!='\0' ){
            temp = 0; // svuoto temp
            temp = (int)otherplayers[in]-'0'; // lo salvo

            // solo se dopo c'è un altro numero faccio questo
            if(otherplayers[in+1]!='?' && otherplayers[in+1]!='&' && otherplayers[in+1]!='\0'){
                temp = 0; // svuoto temp
                temp = (int)otherplayers[in+1]-'0'; // lo salvo
                temp = temp+10; // lo incremento di 10
                in++; // in questo caso avanzo di uno (me ne fotto del primo)
            }

            // a questo punto in temp ho il numero giusto (1 cifra o 2 cifre)
            // dopo [in] c'è sicuro un carattere

            if(otherplayers[in+1]=='?' || otherplayers[in+1]=='&' || otherplayers[in+1]=='\0'){
                nem[indNem]=temp;
                indNem++;
            }
        }
    }
} // fine for

```

server.c

Gestione dei segnali

I segnali gestiti in questo progetto sono i seguenti:

```
signal(SIGINT,handlerCtrlC);  
signal(SIGQUIT,SIG_IGN);  
signal(SIGHUP,SIG_IGN);  
signal(SIGSTOP,SIG_IGN);  
signal(SIGTERM,SIG_IGN);  
signal(SIGABRT,SIG_IGN);  
signal(SIGTSTP,SIG_IGN);  
signal(SIGPIPE,handlerpipe);
```

SIGINT : l'utente interrompe un processo, tipicamente con la combinazione *ctrl+c*

SIGQUIT: chiusura del terminale legato al processo

SIGHUP: chiusura del terminale che controlla tutti i processi

SIGSTOP: segnale di stop

SIGTERM: richiede la terminazione di un processo

SIGABRT: segnale di abort, simile a sigint

SIGTSTP: terminazione del terminale, combinazione di tasti *ctrl+z*

SIGPIPE: un processo prova a leggere da una pipe chiusa, o legata ad un processo terminato.

Gli errori gestiti sono SIGINT e SIGPIPE, in questo modo:

```
void handlerpipe(int x)
{
    void *status;
    char buffer[5000];
    struct Utente *Utente_local=NULL;
    time_t ora;
    pthread_mutex_lock(&mymutex); //blocco il mutex
    Utente_local=getNode(pthread_self(),Ut);
    if(Mappa[Utente_local->x][Utente_local->y]=='U'){
        Mappa[Utente_local->x][Utente_local->y]=' ';
    }
    pthread_mutex_unlock(&mymutex);
    if(Utente_local){
        ora=time(NULL);
        sprintf(buffer,"\t%s\tchiusura anomala\t%s\n",Utente_local->nickname,asctime(localtime(&ora)));
        pthread_mutex_lock(&mymutex); //sblocco il mutex
        Ut=eliminaUtente(Ut,Utente_local->nickname);
        pthread_mutex_unlock(&mymutex);
    }
    pthread_exit(status);
}

//
void handlerCtrlC(int x){
    void *status;
    pthread_mutex_lock(&mymutex);
    struct Utente *local=Ut;
    pthread_mutex_unlock(&mymutex);
    while(local!=NULL){
        pthread_exit(&local->pid);
        local=local->next;
    }
    pthread_exit(status);
}
```

Handlerpipe trova il client che ha generato l'errore attraverso la funzione *getnode*, successivamente resetta il punto della mappa in cui si trovava l'utente prima di essere deallocato.

HandlerCtrlC dealloca tutta la memoria dinamica e fa sì che il server si spenga.

4.Codice sorgente:

Server.c

```
//SERVER.C

//librerie
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <sys/syscall.h>
#include <pthread.h>
#include <time.h>
#include <signal.h>
//colori
#define CYANO  "\x1B[1;36m"
#define YELLOW "\x1B[1;33m"
#define RED    "\x1b[1;31m"
#define GREEN  "\x1b[1;32m"
#define RESET  "\x1b[0m"
#define BLU    "\x1B[1;34m"
//dichiarazione del tempo Totale
#define tempoTotale 120

//struttura dati dell'utente
struct Utente{
    char nickname[512];
    pid_t pid;
    int x;
    int y;
    struct Utente *next;
};

//puntatori al primo elemento delle strutture dati
struct Utente *Ut=NULL;
```



```

//funzioni Thread
void *menu(void *ptr);
void *creaMappa(void *ptr);

//funzioni generiche int(booleane)
int checkLogin(char *,int);
int controllaNome(char *,int);
int giaOnline(char *,struct Utente *);

//funzioni generiche void
void error(char *);
void stampaLog(int ,char *);
void dataCorrente();
void getNickname(char *flag,char *);
void sendSocket(int ,char *,int );
void gioco(int conn,char *);
void stampaMappa(int fd,char Mappa[][20]);
void sig_handler(int);
void handlerpipe(int x);
void handlerCtrlC(int x);

//funzioni per la lista di utenti connessi
struct Utente *eliminaUtente(struct Utente *,char *);
void deallocaUtente(struct Utente *);
void aggiungiUtente(struct Utente **top,char *,pid_t);
void aggiungiPosizione(struct Utente *top,char *nickname,int i,int j);

//Variabili Globali
pthread_mutex_t mymutex=PTHREAD_MUTEX_INITIALIZER;
char Mappa[20][20];
int fdUtente;
int fdlog;
int utentiConnessi=0;
int MappaCreata=0;
int tempoGioco=0;
int nonvittoria=0;
int UtentiOnline=0;
int idGioco=0;

//inizio main
int main(int argc, char const *argv[]){

    int sock, Connesso;
    char datiRicevuti[1024];

```

```

int bytesRicevuti=0;
struct sockaddr_in server_addr,client_addr;
int sockSize,i=0,j=0;

fdUtente=open("utenti.txt", O_RDWR | O_CREAT |O_APPEND ,S_IRWXU);
fdlog=open("log.txt",O_WRONLY | O_CREAT | O_APPEND,S_IRWXU);

pthread_t tred;
pthread_t threadServer;
char data[256];

signal(SIGINT,handlerCtrlC);
signal(SIGQUIT,SIG_IGN);
signal(SIGHUP,SIG_IGN);
signal(SIGSTOP,SIG_IGN);
signal(SIGTERM,SIG_IGN);
signal(SIGABRT,SIG_IGN);
signal(SIGTSTP,SIG_IGN);
signal(SIGPIPE,handlerpipe);

if(fdUtente<0 || fdlog<0){
    error("Errore Permessi FILE\n");
}

if(argc!=2){
    error("Errore Inserire La Porta : Man D'Uso ./server 5xxx\n");
}

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    error("Socket\n");
}

server_addr.sin_family = AF_INET; //accetta un formato host + porta
server_addr.sin_port = htons(atoi(argv[1])); //porta
server_addr.sin_addr.s_addr = INADDR_ANY; //indirizzo

//bind assegna indirizzo di socket locale alla remota
if (bind(sock, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1) {
    error("Unable to bind\n");
}

sockSize = sizeof(struct sockaddr_in);

//listen (socket , lunghezza coda di connessioni)
if (listen(sock, 5) == -1) {

```

```

    error("Listen\n");
}

dataCorrente(data);
bytesRicevuti=sprintf(datiRicevuti,"il server è up sulla %s port in Data:
%s\n",argv[1],data);
stampaLog(fdlog,datiRicevuti);
signal(SIGPIPE,handlerpipe);

while(1){

    if((Connesso=accept(sock, (struct sockaddr *)&client_addr,&sockSize))==-1){
        error("Connessione rifiutata\n");
    }
    utentiConnessi++;

    if((pthread_create(&tred,NULL,menu,(void*)&Connesso)) < 0){
        error("pthread_create() Fallita\n");
    }
    pthread_detach(tred);
    sprintf(datiRicevuti,"Nuova Connessione Da : (%s,
%d)\n",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
    stampaLog(fdlog,datiRicevuti);
}

close(fdUtente);
close(sock);
close(fdlog);
deallocaUtente(Ut);
return 0;
}

```

```

void *menu(void *ptr){

    char Addnickname[512],messaggio[512],scelta[1024],data[256];
    int choosed=1;
    int *connect=(int*)ptr;
    int conn=*connect;
    int nickname=0;
    int fdUtente_loc;
    struct Utente *tmp=NULL;
    int fdlog_loc, Online=0;
    int i=0;
    pthread_t threadServer;

```

```
signal(SIGPIPE,handlerpipe);
pthread_mutex_lock(&mymutex);//blocca il mutex
fdUtente_loc=fdUtente;
fdlog_loc=fdlog;
pthread_mutex_unlock(&mymutex);//sblocca il mutex
```

```
signal(SIGINT,handlerCtrlC);
signal(SIGQUIT,SIG_IGN);
signal(SIGHUP,SIG_IGN);
signal(SIGSTOP,SIG_IGN);
signal(SIGTERM,SIG_IGN);
signal(SIGABRT,SIG_IGN);
signal(SIGTSTP,SIG_IGN);
signal(SIGPIPE,handlerpipe);
```

```
do{
    for(i=0; i<1024; i++){
        scelta[i]='\0';
    }
    if(nickname=read(conn,scelta,1024)<0){
        error("Errore read\n");
    }
}
```

```
if(scelta[0]=='a'){
```

```
    scelta[nickname]='\0';
```

```
    if(controllaNome(scelta+1,fdUtente_loc)){
```

```
        stampaLog(fdUtente_loc,scelta+1);
```

```
        dataCorrente(data);
```

```
        sprintf(messaggio,"Nuovo utente registrato con %s in data : %s\n",scelta+1,data);
```

```
        stampaLog(fdlog_loc,messaggio);
```

```
        sendSocket(conn,"y",1);
```

```
    }else{
```

```
        dataCorrente(data);
```

```
        sprintf(messaggio,"Impossibile registrare l'Utente con %s in data :
```

```
%s\n",scelta+1,data);
```

```
        stampaLog(fdlog_loc,messaggio);
```

```
        sendSocket(conn,"n",1);
```

```
    }
```

```
}
```

```
else if(scelta[0]=='b'){
```

```
    scelta[nickname]='\0';
```

```

getNickname(Addnickname,scelta+1);
pthread_mutex_lock(&mymutex);//blocca il mutex
Online=giaOnline(Addnickname,Ut);
struct Utente **tmp=&Ut;
pthread_mutex_unlock(&mymutex);//sblocca il mutex

if(Online==1){
    dataCorrente(data);
    sprintf(messaggio,"Impossibile effettuare il login l'Utente è già loggato con %s in
data : %s\n",scelta+1,data);
    stampaLog(fdlog_loc,messaggio);
    sendSocket(conn,"g",1);
    Online=0;
}
else{
    dataCorrente(data);
    sprintf(messaggio,"L'utente %s si e' connesso in data : %s\n",Addnickname,data);
    stampaLog(fdlog_loc,messaggio);
    if(!checkLogin(scelta+1,fdUtente_loc)){
        sendSocket(conn,"y",1);
        pthread_mutex_lock(&mymutex);//blocca il mutex
        aggiungiUtente(tmp,Addnickname,pthread_self());
        pthread_mutex_unlock(&mymutex);//sblocca il mutex
        if(MappaCreata==0){
            if(pthread_create(&threadServer,NULL,creaMappa,NULL) < 0){
                error("Errore Thread\n");
            }
        }
        gioco(conn,Addnickname);
    }else{
        sendSocket(conn,"n",1);
    }
}
}
}
else if(scelta[0]=='c'){
    int fileLog_loc2=fdlog;
    dataCorrente(data);
    pthread_mutex_lock(&mymutex);//blocca il mutex
    char messaggio[256];
    sprintf(messaggio,"Utente non loggato disconnesso in Data %s\n",data);
    stampaLog(fileLog_loc2,messaggio);
    utentiConnessi--;
    pthread_mutex_unlock(&mymutex);//sblocca il mutex
    choosed=0;
} else if(scelta[0]=='l'){

```

```

pthread_mutex_lock(&mymutex);//blocca il mutex
int fileLog_loc2=fdlog;
dataCorrente(data);
char messaggio[256];
sprintf(messaggio,"Utente non loggato disconnesso in maniera anomala in Data
%s\n",data);
    stampaLog(fileLog_loc2,messaggio);
    utentiConnessi--;
    pthread_mutex_unlock(&mymutex);//sblocca il mutex
    choosed=0;

} else {
    pthread_mutex_lock(&mymutex);//blocca il mutex
    int fileLog_loc3=fdlog;
    dataCorrente(data);
    sprintf(messaggio,"Errore di comunicazione con client in Data %s\n",data);
    stampaLog(fileLog_loc3,messaggio);
    pthread_mutex_unlock(&mymutex);//sblocca il mutex
    choosed=0;
}
} while(choosed==1);

pthread_exit(NULL);
}

void sendSocket(int conn,char *mess,int nbytes){
    if(write(conn,mess,nbytes)<0){
        error("Errore write\n");
    }
}

// _____

void error(char *messaggio){
    perror(messaggio);
    pthread_mutex_lock(&mymutex);//blocco il mutex in scrittura del file
    int fdlog_loc=fdlog;
    pthread_mutex_unlock(&mymutex);//sblocco il mutex in scrittura del file

    stampaLog(fdlog_loc,messaggio);
    exit(1);
}

```

```
// _____  
_____
```

```
void stampaLog(int fd,char *messaggio){  
    if(write(fd,messaggio,strlen(messaggio))<0){  
        error("Permessi Write\n");  
    }  
}
```

```
// _____  
_____
```

```
void dataCorrente(char *dataT)  
{  
    int gm,m,a,gs,h,min,s;  
    time_t data;  
    struct tm * leggibile = NULL;  
    time (&data);  
  
    leggibile = localtime (&data);  
    gm=leggibile->tm_mday;  
    m=leggibile->tm_mon +1;  
    a=leggibile->tm_year+1900;  
    gs=leggibile->tm_wday+1; // 1 = Domenica - 7 = Sabato  
    h=leggibile->tm_hour;  
    min=leggibile->tm_min;  
    s=leggibile->tm_sec;  
    sprintf(dataT,"%d/%d/%d Ora :(%d:%d:%d)",gm,m,a,h,min,s);  
}
```

```
// _____  
_____
```

```
int controllaNome(char *nickname,int fd)  
{  
    char flag[1024], file[1024], nome[1024];  
    int i=0,nbytes=0;  
    off_t currpos;  
  
    currpos = lseek(fd, 0, SEEK_SET);  
    getNickname(flag,nickname);
```

```

while(read(fd, &file[i], 1) == 1){
    if(file[i]=='\n' || file[i]==0x0){
        file[i]='\0';
        (nome,file);
        if(strcmp(flag,nome)==0){
            return 0;
        }
        i=0;
        continue;
    }
    i++;
}
return 1;
}

```

```

// _____
_____

```

```

void getNickname(char *flag,char *nickname)
{
    int i=0;

    while(nickname[i]!=':'){
        flag[i]=nickname[i];
        i++;
    }
    flag[i]='\0';
}

```

```

// _____
_____

```

```

int checkLogin(char *nickname, int fd)
{
    char flag[1024], file[1024], nome[1024];
    int i=0,nbytes=0;
    off_t currpos;

    currpos = lseek(fd, 0, SEEK_SET);
    while(read(fd, &file[i], 1) == 1){
        if(file[i]=='\n' || file[i]==0x0){
            file[i]='\n';
            file[i+1]='\0';

```



```

        if(strcmp(file,nickname)==0){
            return 0;
        }
        i=0;
        continue;
    }
    i++;
}
return 1;
}

```

```

// _____
_____

```

```

void gioco(int conn, char *nickname)
{

    char coordinateClient[3], nuovaPosizione[3];
    srand(time(NULL));
    int i=0, j=0, flag=1, q=0, k=0, posizionato=0, Ric_messaggio=0, controlloId=0;
    char messaggio[1000];
    char lista[5000], data[256];
    int vittoria=0;
    pthread_t threadServer;
    int idGioco_client;
    pthread_mutex_lock(&mymutex);//blocco il mutex
    UtentiOnline++;
    pthread_mutex_unlock(&mymutex);//sblocco il mutex

    nuovaPosizione[0]='\0';
    nuovaPosizione[1]='\0';
    nuovaPosizione[2]='\0';

    do{

        i=rand()%19;
        j=0;
        if(Mappa[i][j]==' '){
            pthread_mutex_lock(&mymutex);//blocco il mutex
            Mappa[i][j]='U';
            pthread_mutex_unlock(&mymutex);//blocco il mutex
            posizionato=1;
        }
    }while(posizionato==0);
    coordinateClient[0]=i;

```

```

coordinateClient[1]=j;
coordinateClient[2]='U';

//inviare qui l'id partita
if(write(conn,&idGioco,sizeof(int))<0){
    error("Errore write\n");
}
//fine invio id partita

sendSocket(conn,coordinateClient,3);
signal(SIGINT,handlerCtrlC);
signal(SIGQUIT,SIG_IGN);
signal(SIGHUP,SIG_IGN);
signal(SIGSTOP,SIG_IGN);
signal(SIGTERM,SIG_IGN);
signal(SIGABRT,SIG_IGN);
signal(SIGTSTP,SIG_IGN);
signal(SIGPIPE,handlerpipe);
do{

    pthread_mutex_lock(&mymutex);//blocco il mutex
    aggiungiPosizione(Ut,nickname,coordinateClient[0],coordinateClient[1]);
    pthread_mutex_unlock(&mymutex);//sblocco il mutex

    nuovaPosizione[0]=0;
    if(read(conn,nuovaPosizione,3)<0){
        flag=0;
    }

    else{

        switch(nuovaPosizione[0]){
            case 'w':

                if(read(conn,&idGioco_client,sizeof(int))<0){
                    error("Errore read");
                }
                if(tempoGioco==tempoTotale){
                    pthread_mutex_lock(&mymutex);//blocco il mutex
                    struct Utente *tmp=Ut;
                    pthread_mutex_unlock(&mymutex);//sblocco il mutex

                    nonvittoria=1;
                    flag=0;
                }
                else if(idGioco!=idGioco_client){

```

```

        sendSocket(conn,"f",1);
        controlloId=1;
        flag=0;
    }else{
        if((nuovaPosizione[1]-1)<0){
            coordinateClient[0]=nuovaPosizione[1];
            coordinateClient[1]=nuovaPosizione[2];
        }
        else{
            if(Mappa[nuovaPosizione[1]-1][nuovaPosizione[2]]=='X'){
                coordinateClient[0]=nuovaPosizione[1];
                coordinateClient[1]=nuovaPosizione[2];
                coordinateClient[2]='X';
                pthread_mutex_lock(&mymutex);//blocca il mutex
                Ut=eliminaUtente(Ut,nickname);
                pthread_mutex_unlock(&mymutex);//sblocca il mutex
                flag=0;

            }
            else if(Mappa[nuovaPosizione[1]-1][nuovaPosizione[2]]=='U'){
                coordinateClient[0]=nuovaPosizione[1];
                coordinateClient[1]=nuovaPosizione[2];
                coordinateClient[2]='U';
            }

            else{
                pthread_mutex_lock(&mymutex);//blocco il mutex
                if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=='U'){
                    Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=' ';
                }
                coordinateClient[0]=nuovaPosizione[1]-1;
                coordinateClient[1]=nuovaPosizione[2];
                coordinateClient[2]=' ';
                Mappa[nuovaPosizione[1]-1][nuovaPosizione[2]]='U';
                pthread_mutex_unlock(&mymutex);//sblocco il mutex
            }
        }
    }
}
if(nonvittoria==1){
    sendSocket(conn,"f",1);
    flag=0;
}else{
    if(controlloId==0){
        sendSocket(conn,coordinateClient,3);
    }
}

```

```
}  
break;
```

case 's':

```
if(read(conn,&idGioco_client,sizeof(int))<0){  
    error("Errore read");  
}  
if(idGioco!=idGioco_client){  
    sendSocket(conn,"f",1);  
    controlloId=1;  
    flag=0;  
}  
else if(tempoGioco==tempoTotale){  
    pthread_mutex_lock(&mymutex);//blocco il mutex  
    struct Utente *tmp=Ut;  
    pthread_mutex_unlock(&mymutex);//sblocco il mutex  
    nonvittoria=1;  
    flag=0;  
}else{  
    if((nuovaPosizione[1]+1)>19){  
        coordinateClient[0]=nuovaPosizione[1];  
        coordinateClient[1]=nuovaPosizione[2];  
    }  
    else{  
        if(Mappa[nuovaPosizione[1]+1][nuovaPosizione[2]]=='X'){  
            coordinateClient[0]=nuovaPosizione[1];  
            coordinateClient[1]=nuovaPosizione[2];  
            coordinateClient[2]='X';  
            pthread_mutex_lock(&mymutex);//blocca il mutex  
            Ut=eliminaUtente(Ut,nickname);  
            pthread_mutex_unlock(&mymutex);//sblocca il mutex  
            flag=0;  
        }  
        else if(Mappa[nuovaPosizione[1]+1][nuovaPosizione[2]]=='U'){  
            coordinateClient[0]=nuovaPosizione[1];  
            coordinateClient[1]=nuovaPosizione[2];  
            coordinateClient[2]='U';  
        }  
    }  
  
    else{  
        pthread_mutex_lock(&mymutex);//blocco il mutex  
        if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=='U'){  
            Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=' ';  
        }  
    }  
}
```

```

        coordinateClient[0]=nuovaPosizione[1]+1;
        coordinateClient[1]=nuovaPosizione[2];
        coordinateClient[2]=' ';
        Mappa[nuovaPosizione[1]+1][nuovaPosizione[2]]='U';
        pthread_mutex_unlock(&mymutex);//sblocco il mutex
    }
}
}
if(nonvittoria==1){
    sendSocket(conn,"f",1);
    flag=0;
}else{
    if(controllId==0){
        sendSocket(conn,coordinateClient,3);

    }
}
break;

case 'a':

if(read(conn,&idGioco_client,sizeof(int))<0){
    error("Errore read");
}
if(idGioco!=idGioco_client){
    sendSocket(conn,"f",1);
    controllId=1;
    flag=0;
}
else if(tempoGioco==tempoTotale){
    pthread_mutex_lock(&mymutex);//blocco il mutex
    struct Utente *tmp=Ut;
    pthread_mutex_unlock(&mymutex);//sblocco il mutex
    nonvittoria=1;
    flag=0;
}else{
    if((nuovaPosizione[2]-1)<0){
        coordinateClient[0]=nuovaPosizione[1];
        coordinateClient[1]=nuovaPosizione[2];
    }
    else{
        if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]-1]=='X'){
            coordinateClient[0]=nuovaPosizione[1];
            coordinateClient[1]=nuovaPosizione[2];
            coordinateClient[2]='X';
            pthread_mutex_lock(&mymutex);//blocca il mutex

```

```

        Ut=eliminaUtente(Ut,nickname);
        pthread_mutex_unlock(&mymutex);//sblocca il mutex
        flag=0;
    }
    else if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]-1]=='U'){
        coordinateClient[0]=nuovaPosizione[1];
        coordinateClient[1]=nuovaPosizione[2];
        coordinateClient[2]='U';
    }

    else{
        pthread_mutex_lock(&mymutex);//blocco il mutex
        if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=='U'){
            Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=' ';
        }
        coordinateClient[0]=nuovaPosizione[1];
        coordinateClient[1]=nuovaPosizione[2]-1;
        coordinateClient[2]=' ';
        Mappa[nuovaPosizione[1]][nuovaPosizione[2]-1]='U';
        pthread_mutex_unlock(&mymutex);//sblocco il mutex
    }
}
}
if(nonvittoria==1){
    sendSocket(conn,"f",1);
    flag=0;
} else {
    if(controlloId==0){
        sendSocket(conn,coordinateClient,3);
    }
}
}
break;

case 'd':

    if(read(conn,&idGioco_client,sizeof(int))<0){
        error("Errore read");
    }
    if(idGioco!=idGioco_client){
        sendSocket(conn,"f",1);
        controlloId=1;
        flag=0;
    }
    else if(tempoGioco==tempoTotale){

```

```

pthread_mutex_lock(&mymutex);
struct Utente *tmp=Ut;
pthread_mutex_unlock(&mymutex);

nonvittoria=1;
flag=0;
} else {
    if((nuovaPosizione[2]+1)>19){
        coordinateClient[0]=nuovaPosizione[1];
        coordinateClient[1]=nuovaPosizione[2];
    }
    if(nuovaPosizione[2]==19){
        vittoria=1;
    }
    else {
        if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]+1]=='X'){
            coordinateClient[0]=nuovaPosizione[1];
            coordinateClient[1]=nuovaPosizione[2];
            coordinateClient[2]='X';
            pthread_mutex_lock(&mymutex); //blocca il mutex
            Ut=eliminaUtente(Ut,nickname);
            pthread_mutex_unlock(&mymutex); //sblocca il mutex
            flag=0;
        }
        else if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]+1]=='U'){
            coordinateClient[0]=nuovaPosizione[1];
            coordinateClient[1]=nuovaPosizione[2];
            coordinateClient[2]='U';
        }

        else {
            pthread_mutex_lock(&mymutex); //blocco il mutex
            if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=='U'){
                Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=' ';
            }
            coordinateClient[0]=nuovaPosizione[1];
            coordinateClient[1]=nuovaPosizione[2]+1;
            coordinateClient[2]=' ';
            Mappa[nuovaPosizione[1]][nuovaPosizione[2]+1]='U';
            pthread_mutex_unlock(&mymutex); //sblocco il mutex
        }
    }
}
if(vittoria==1){
    sendSocket(conn,"v",1);
    pthread_mutex_lock(&mymutex);

```

```

        deallocaUtente(Ut);
        Ut=NULL;
        flag=0;
        pthread_mutex_unlock(&mymutex);
    }else if(nonvittoria==1){
        sendSocket(conn,"f",1);
        flag=0;
    }else{
        if(controlloId==0){
            sendSocket(conn,coordinateClient,3);

        }
    }
    break;

```

case 'c':

```

        dataCorrente(data);
        pthread_mutex_lock(&mymutex); //blocca il mutex
        int fileLog_loc=fdlog;
        if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=='U'){
            Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=' ';
        }
        Ut=eliminaUtente(Ut,nickname);
        char messaggio[256];
        sprintf(messaggio,"Utente %s Disconnesso in Data %s\n",nickname,data);
        stampaLog(fileLog_loc,messaggio);
        pthread_mutex_unlock(&mymutex); //sblocca il mutex
        flag=0;
        break;

```

case 'i':

```

        pthread_mutex_lock(&mymutex); //blocca il mutex
        struct Utente *tmp=Ut;
        pthread_mutex_unlock(&mymutex); //sblocca il mutex
        for(k=0;k<5000;k++){
            lista[k]='\0';
        }
        strcat(lista,GREEN"Utenti online: "RESET);
        while(tmp!=NULL){
            char coordinateLoc[50];
            sprintf(coordinateLoc,"%d[%d]",tmp->x,tmp->y);
            strcat(lista,GREEN"\nUtente "RESET);
            strcat(lista,tmp->nickname);
            strcat(lista,GREEN" in posizione "RESET);
            strcat(lista,coordinateLoc);
        }

```



```

        strcat(lista, "\n");
        tmp=tmp->next;
    }
    sendSocket(conn, lista, strlen(lista));
    break;

case 't':
    pthread_mutex_lock(&mymutex); //blocca il mutex
    int tempo_loc=tempoGioco;
    pthread_mutex_unlock(&mymutex); //sblocca il mutex
    if(write(conn, &tempo_loc, sizeof(int))<0){
        error("Errore write\n");
    }
    break;

case 'k':
    pthread_mutex_lock(&mymutex); //blocca il mutex
    struct Utente *tmp2=Ut;
    pthread_mutex_unlock(&mymutex); //sblocca il mutex
    char lista2[5000];
    for(k=0; k<5000; k++){
        lista2[k]='\0';
    }
    while(tmp2!=NULL){
        char coordinateLoc[50];
        sprintf(coordinateLoc, "%d?%d", tmp2->x, tmp2->y);
        strcat(lista2, coordinateLoc);
        strcat(lista2, "&");
        tmp2=tmp2->next;
    }
    lista2[strlen(lista2)-1]='\0';
    sendSocket(conn, lista2, strlen(lista2));
    break;

default :
    flag=0;
    break;
}
}
} while(flag==1);

pthread_mutex_lock(&mymutex);
if(Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=='U'){
    Mappa[nuovaPosizione[1]][nuovaPosizione[2]]=' ';
}

```

```

    }
    UtentiOnline--;
    pthread_mutex_unlock(&mymutex);

    Ut=eliminaUtente(Ut,nickname);

    if(vittoria==1){
        if(pthread_create(&threadServer,NULL,creaMappa,NULL) < 0){
            error("Errore Thread\n");
        }
    }
}

```

// _____

```

void stampaMappa(int fd,char Mappa[][20])
{
    int i=0, j=0,z=0;
    char messaggio[20]="Debug\0";
    write(fd,"-----\n",22);
    for(i=0; i<20; i++){
        write(fd,"|",1);
        for(j=0; j<20; j++){
            write(fd,&Mappa[i][j],1);
        }
        write(fd,"|\n",2);
    }
    write(fd,"-----\n",22);
}

```

// _____

```

void *creaMappa(void *ptr)
{
    int i=0, j=0, k=0;
    int fdlog_loc;
    int posizionato=0;
    int cordinata1=0,cordinata2=0;
    char data[50], nuovaMappa[50];
    srand(time(NULL));

```

```

time_t start=0,end=0;
char messId[50];

pthread_mutex_lock(&mymutex);
idGioco++; //aumenta id partita
sprintf(messId,"Inizia la partita numero ( %d )\n",idGioco);
nonvittoria=0;
fdlog_loc=fdlog;
pthread_mutex_unlock(&mymutex);

//inizializzazione della matrice
for(i=0; i<20; i++){
    for(j=0; j<20; j++){
        Mappa[i][j]=' ';
    }
}
//posizionamento delle bombe
for(i=0; i<8; i++){
    cordinata1=1+rand()%18;
    cordinata2=1+rand()%18;
    if(Mappa[cordinata1][cordinata2]==' '){
        Mappa[cordinata1][cordinata2]='X';
    }
}
MappaCreata=1;
sprintf(nuovaMappa,"Nuova mappa generata in data %s\n",data);
stampaLog(fdlog_loc,nuovaMappa);
stampaMappa(fdlog_loc,Mappa);
dataCorrente(data);
tempoGioco=0;
do{
    sleep(1);
    tempoGioco++;
}while(tempoGioco<tempoTotale);

MappaCreata=0;
pthread_exit(NULL);
}

```

//

```

void aggiungiUtente(struct Utente **top,char *nickname,pid_t pidl){

    struct Utente *tmp=(struct Utente *)calloc(1,sizeof(struct Utente));

```

```
strcpy(tmp->nickname,nickname);
tmp->pid=pidl;
tmp->next=(*top);
(*top)=tmp;
}
```

```
//
```

```
struct Utente *eliminaUtente(struct Utente *top,char *nickname){
    if(top!=NULL){
        if(strcmp(top->nickname,nickname)==0){
            struct Utente *tmp=NULL;
            tmp=top;
            top=top->next;
            free(tmp);
        }else{
            top->next=eliminaUtente(top->next,nickname);
        }
    }
    return top;
}
```

```
//
```

```
int giaOnline(char *nickname ,struct Utente *top){
    int flag=0;
    if(top!=NULL){
        if(strcmp(top->nickname,nickname)==0){
            flag=1;
        }else{
            flag=giaOnline(nickname,top->next);
        }
    }
    return flag;
}
```

```
//
```

```
void deallocaUtente(struct Utente *top){
```

```
    if(top!=NULL){
        deallocaUtente(top->next);
        free(top);
    }
}
```

```
//
```

```
void aggiungiPosizione(struct Utente *top,char *nickname,int i,int j){
    if(top!=NULL){
        if(strcmp(top->nickname,nickname)==0){
            top->x=i;
            top->y=j;
        }else{
            aggiungiPosizione(top->next,nickname,i,j);
        }
    }
}
```

```
//
```

```
struct Utente *getNode(pid_t pid,struct Utente *top){
    struct Utente *flag=0;
    if(top!=NULL){
        if(top->pid==pid){
            flag=top;
        }else{
            flag=getNode(pid,top->next);
        }
    }
    return flag;
}
```

```
//
```

```
void handlerpipe(int x)
{
    void *status;
    char buffer[5000];
}
```

```

struct Utente *Utente_local=NULL;
time_t ora;
pthread_mutex_lock(&mymutex); //blocco il mutex
Utente_local=getNode(pthread_self(),Ut);
if(Mappa[Utente_local->x][Utente_local->y]=='U'){
    Mappa[Utente_local->x][Utente_local->y]=' ';
}
pthread_mutex_unlock(&mymutex);
if(Utente_local){
    ora=time(NULL);
    sprintf(buffer,"%t%s\tchiusura anomala\t%s\n",Utente_local-
>nickname,asctime(localtime(&ora)));
    pthread_mutex_lock(&mymutex); //sblocco il mutex
    Ut=eliminaUtente(Ut,Utente_local->nickname);
    pthread_mutex_unlock(&mymutex);
}
pthread_exit(status);
}

```

//

```

void handlerCtrlC(int x){
    void *status;
    pthread_mutex_lock(&mymutex);
    struct Utente *local=Ut;
    pthread_mutex_unlock(&mymutex);
    while(local!=NULL){
        pthread_exit(&local->pid);
        local=local->next;
    }
    pthread_exit(status);
}

```

Client.c

//client.c//

```

//librerie
#include <sys/socket.h>
#include <sys/types.h>

```

```

#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <time.h>
#include <signal.h>
//Colori
#define CYANO  "\x1B[1;36m"
#define YELLOW "\x1B[1;33m"
#define RED    "\x1b[1;31m"
#define GREEN  "\x1b[1;32m"
#define RESET  "\x1b[0m"
#define BLU    "\x1B[1;34m"
#define tempoTot 120


//funzioni

void stampa(char *messaggio);
//stampa a video il frontend del menu
void menu();
//pulisce il buffer
void pulisciBuffer();
//esegue una perror sul messaggio
void error(char *messaggio);
//stampa la mappa di gioco con giocatori
void stampaMappa(char A[][20],int,int,int,char *);
//funzione principale del gioco
void gioco(int sock);
//invia una socket al server
void sendSocket(int conn,char *mess,int nbytes);
//funzioni sui segnali
void sighandler(int segnale_ricevuto);
void sighandler1(int segnale_ricevuto);
void sighandler2(int segnale_ricevuto);
void sighandler3(int segnale_ricevuto);
void sighandler4(int segnale_ricevuto);
//stampa a video la vittoria in caso di raggiungimento traguardo
void stampaVittoria();
//stampa a video la sconfitta in caso di bomba
void stampaSconfitta();

```

```
//pausa
void pauseT();
```

```
//variabili globali
```

```
//variabile che determina
int giocato=0;
//variabile che stabilisce se sono in partita
int inGioco=0;
int connessioneFlag=0;
int flagPosizione1=0;
int flagPosizione2=0;
```

```
int main(int argc, char const *argv[])
{
    int uscita=0; //variabile per il logout
    int sock; //fd della socket
    int i=0, j=0;
    int bytesnickname=0, byteword=0;
    char Login[1024];
    char risposta;
    char scelta; //scelta di selezione
    char Nickname[256], Password[256], Log[512];
    struct hostent *host;
    struct sockaddr_in server_addr;

    //controllo input all'avvio del client
    if(argc!=3){
        error(RED"Error : Man = ./client ip(xxx.xxx.xxx) port(5XXX)\n"RESET);
    }

    host = gethostbyname(argv[1]);

    if((sock=socket(AF_INET,SOCK_STREAM,0))==-1){
        error(RED"ERROR : Socket\n"RESET);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(atoi(argv[2]));
    server_addr.sin_addr = *((struct in_addr *)host->h_addr);
    bzero(&(server_addr.sin_zero),8);
```



```
if(connect(sock,(struct sockaddr *)&server_addr,sizeof(struct sockaddr)) == -1){  
    error(RED"ERROR : Connect\n"RESET);  
}
```

```
conessioneFlag=sock;  
system("clear");
```

```
//inizio frontend menu registrazione,login,logout  
do{  
    if(giocato==1){  
        sleep(1);  
        sprintf(Login,"b%s:%s\n",Nickname>Password);  
  
        if(write(sock,Login,strlen(Login))<0){  
            error("Errore WRITE\n");  
        }  
  
        if(read(sock,&risposta,1)<0){  
            error("Error Read\n");  
        }  
        inGioco=1;  
        gioco(sock);  
    }  
}
```

```
else{  
    //scelta tasti di input  
    scelta=0;  
  
    menu();  
  
    signal(SIGINT,sighandler1);  
    signal(SIGQUIT,sighandler1);  
    signal(SIGHUP,sighandler2);  
    signal(SIGSTOP,sighandler2);  
    signal(SIGTERM,sighandler2);  
    signal(SIGABRT,sighandler2);  
    signal(SIGTSTP,sighandler2);  
    signal(SIGPIPE,sighandler4);  
  
    if(read(1,&scelta,1)<0){  
        error("error write\n");  
    }  
  
    //a) registrazione utente  
    if(scelta=='a'){
```

```

pulisciBuffer();

stampa(BLU"Inserire Nickname e Password\n"RESET);
stampa("Nickname : ");

if((bytesnickname=read(1,Nickname,256))<0){
    error("Errore write\n");
}
Nickname[bytesnickname-1]='\0';

stampa("\nPassword : ");
if((bytepassword=read(1>Password,256))<0){
    error("Errore write\n");
}
Password[bytepassword-1]='\0';

sprintf(Login,"a%s:%s\n",Nickname>Password);

if(write(sock>Login,strlen(Login))<0){
    error("Errore WRITE\n");
}

if(read(sock,&risposta,1)<0){
    error("Error Read\n");
}
switch(risposta){
    case 'y': stampa(GREEN"Registrazione avvenuta con Successo, verrai
indirizzato alla schermata iniziale\n"RESET);pauseT(); system("clear");break;
    case 'n': stampa(RED"Il Nickname è già stato usato, verrai indirizzato alla
schermata iniziale\n"RESET); pauseT(); system("clear"); break;
    default : error("Impossibile comunicare con il server\n"); break;
}
}

//b) login
else if(scelta=='b'){
    pulisciBuffer();

    stampa(BLU"Inserire Nickname e Password Per Il Login\n"RESET);
    stampa("Nickname : ");

    if((bytesnickname=read(1,Nickname,256))<0){
        error("Errore write\n");
    }
    Nickname[bytesnickname-1]='\0';

```

```

        stampa("\nPassword : ");
        if((bytepassword=read(1,Password,256))<0){
            error("Errore write\n");
        }
        Password[bytepassword-1]='\0';

        sprintf(Login,"b%s:%s\n",Nickname,Password);
        if(write(sock,Login,strlen(Login))<0){
            error("Errore WRITE\n");
        }
        if(read(sock,&risposta,1)<0){
            error("Error Read\n");
        }
        switch(risposta){
            case 'y':system("clear"); stampa(GREEN"Adesso puoi iniziare a
giocare!\n"RESET);
                inGioco=1;
                gioco(sock);
                break;

            case 'n': stampa(RED"Login fallito utente non registrato\n"RESET); pauseT();
system("clear");
                break;

            case 'g': stampa(RED"Login fallito utente già Online\n"RESET); pauseT();
system("clear");
                break;

            default : stampa("Impossibile comunicare con il server\n");
                break;
        }
    }

    //c) logout
    else if(scelta=='c'){
        stampa(YELLOW"Chiusura Del Client In Corso\n"RESET);
        sleep(1);

        if(write(sock,"c",1)<0){//comunica al server che è stata inserita una c
            error("error write\n");
        }
        uscita=1;
    }
    else{
        system("clear");
        pulisciBuffer();
    }
}

```

```

        stampa(RED"Error input riprova\n"RESET);
    }
}
}while(!uscita);

close(sock);
pulisciBuffer();
return 0;
}

```

```

// _____
FUNZIONE

```

```

void menu() {
    system("clear");
    stampa(GREEN);
    stampa(" PROGETTO ESAME Lab Sistemi Operativi\nDocente: Alberto Finzi\nAnno_
2018/19"RESET);
    stampa(BLU"\nAlberto Panzera N86002772\nPellecchia Chiara
N86002277\n\n"RESET);
    stampa("_____ \n");
    stampa("  MENU DI GIOCO  \n");
    stampa("-----\n");
    stampa("a) Crea un Account  \n");
    stampa("b) Login          \n");
    stampa("c) Logout           \n");
    stampa("-----\n");
}

```

```

// _____
FUNZIONE

```

```

void stampa(char *messaggio){
    int l=strlen(messaggio);
    if(write(0,messaggio,l)<0){
        error("Error write\n");
    }
}

```

```

// _____
FUNZIONE

```

```

void pulisciBuffer(){
    char ch;
    while((ch=getchar())!='\n');
}

```

```
// _____
FUNZIONE
void error(char *messaggio){
    perror(messaggio);
    exit(1);
}

// _____
FUNZIONE
void gioco(int sock){

    int i=0, j=0;
    int k=0, vittoria=0;
    char nuovaPosizione[3], miaPosizione[3], y=0;
    char Mappa[20][20], lista[5000], prova;
    char messaggio[256];
    int id_corrente, id_server;
    int tempo=0;
    char tempo_loc[256];

    //posiziona "-" per ogni posizione della mappa
    for(i=0; i<20; i++){
        for(j=0; j<20; j++){
            Mappa[i][j]='-';
        }
    }

    //leggere qui il primo id
    if(read(sock, &id_corrente, sizeof(int))<0){
        error("errore read\n");
    }

    //fine lettura primo id
    if(read(sock, nuovaPosizione, 3)<0){
        error("errore read\n");
    }

    i=nuovaPosizione[0];
    j=nuovaPosizione[1];
    Mappa[i][j]=nuovaPosizione[2];
    system("clear");
    stampaMappa(Mappa, i, j, id_corrente, NULL);

    do{
        signal(SIGINT, sighandler);
```

```
signal(SIGQUIT,sighandler);
signal(SIGHUP,sighandler3);
signal(SIGSTOP,sighandler3);
signal(SIGTERM,sighandler3);
signal(SIGTSTP,sighandler3);
signal(SIGPIPE,sighandler4);
signal(SIGABRT,sighandler3);
```

```
flagPosizione1=i;
flagPosizione2=j;
```

```
if(read(1,&miaPosizione[0],1)<=0){
    error("error read\n");
}
pulisciBuffer();
```

```
int z=0;
for(z=0; z<256; z++){
    messaggio[z]='\0';
}
```

```
//risposta in base alla posizione
switch(miaPosizione[0]){
```

```
    //w) spostamento in alto
    case 'w':
        miaPosizione[1]=i;
        miaPosizione[2]=j;
        sendSocket(sock,miaPosizione,3);
```

```
    //write id del client verso il server
    if(write(sock,&id_corrente,sizeof(int))<0){
        error("Errore write");
    }
```

```
    //legge la risposta del server
    if(read(sock,nuovaPosizione,3)<0){
        error(RED"Error Read\n"RESET);
        inGioco=0; giocato=0;
    }
```

```
    //Tempo scaduto o vince un altro
    if(nuovaPosizione[0]=='f'){
        system("clear");
        stampa(RED"Partita terminata!\n"RESET);
```

```

    inGioco=0;
    giocato=1;
}

if(nuovaPosizione[2]=='X'){
    stampaSconfitta();
    sleep(1);
    inGioco=0;
    giocato = 0;
}

else{
    Mappa[i][j]='-';
    i=nuovaPosizione[0]; j=nuovaPosizione[1];
    Mappa[i][j]='U';

    for(k=0;k<5000;k++){
        lista[k]='\0';
    }

    sendSocket(sock,"k",1);
    if(read(sock,lista,5000)<0){
        error("errore read\n");
    }
    stampaMappa(Mappa,i,j,id_corrente,lista);
}
break;

```

//s) spostamento in basso

case 's':

```

    miaPosizione[1]=i;
    miaPosizione[2]=j;
    sendSocket(sock,miaPosizione,3);

```

//invio dell id al server

```

    if(write(sock,&id_corrente,sizeof(int))<0){
        error("Errore write");
    }

```

```

    if(read(sock,nuovaPosizione,3)<0){
        error(RED"Error Read\n"RESET);
        inGioco=0; giocato=0;
    }

```

```

    if(nuovaPosizione[0]=='f'){

```

```

    system("clear");
    stampa(RED"Partita terminata!\n"RESET);
    inGioco=0;
    giocato=1;
}

if(nuovaPosizione[2]=='X'){
    stampaSconfitta();
    sleep(1);
    inGioco=0;
    giocato=0;
}

else{
    Mappa[i][j]='-';
    i=nuovaPosizione[0]; j=nuovaPosizione[1];
    Mappa[i][j]='U';

    for(k=0;k<5000;k++){
        lista[k]='\0';
    }

    sendSocket(sock,"k",1);
    if(read(sock,lista,5000)<0){
        error("errore read\n");
    }
    stampaMappa(Mappa,i,j,id_corrente,lista);
}
break;

```

//a) spostamento a sinistra

case 'a':

```

    miaPosizione[1]=i;
    miaPosizione[2]=j;
    sendSocket(sock,miaPosizione,3);

```

//invio dell id al server

```

    if(write(sock,&id_corrente,sizeof(int))<0){
        error("Errore write");
    }

```

```

    if(read(sock,nuovaPosizione,3)<0){
        error(RED"Error Read\n"RESET);
        inGioco=0; giocato=0;
    }

```



```

if(nuovaPosizione[0]=='f'){
    system("clear");
    stampa(RED"Partita terminata!\n"RESET);
    inGioco=0;
    giocato=1;
}

if(nuovaPosizione[2]=='X'){
    stampaSconfitta();
    sleep(1);
    inGioco=0;
    giocato=0;
}

else{
    Mappa[i][j]='-';
    i=nuovaPosizione[0]; j=nuovaPosizione[1];
    Mappa[i][j]='U';

    for(k=0;k<5000;k++){
        lista[k]='\0';
    }

    sendSocket(sock,"k",1);
    if(read(sock,lista,5000)<0){
        error("errore read\n");
    }
    stampaMappa(Mappa,i,j,id_corrente,lista);
}
break;

```

//d) spostamento a destra
case 'd':

```

miaPosizione[1]=i;
miaPosizione[2]=j;
sendSocket(sock,miaPosizione,3);

//invio dell id al server
if(write(sock,&id_corrente,sizeof(int))<0){
    error("Errore write");
}

if(read(sock,nuovaPosizione,3)<0){
    error(RED"Error Read\n"RESET);
}

```

```

        inGioco=0; giocato=0;
    }

    if(nuovaPosizione[0]=='v'){
        stampaVittoria();
        sleep(1);
        inGioco=0;
        vittoria=1;
        giocato=1;
    } else if(nuovaPosizione[0]=='f'){
        system("clear");
        stampa(RED"Partita terminata!\n"RESET);
        inGioco=0;
        giocato=1;
    }

    else if(nuovaPosizione[2]=='X'){
        stampaSconfitta();
        sleep(1);
        inGioco=0;
        giocato=0;
    }
    else{
        Mappa[i][j]='-';
        i=nuovaPosizione[0]; j=nuovaPosizione[1];
        Mappa[i][j]='U';

        for(k=0;k<5000;k++){
            lista[k]='\0';
        }
        sendSocket(sock,"k",1);
        if(read(sock,lista,5000)<0){
            error("errore read\n");
        }
        stampaMappa(Mappa,i,j,id_corrente,lista);
    }
    break;

```

```

//c) logout
case 'c':

```

```

    stampa(YELLOW"Stai per uscire dal gioco e tornare al menu iniziale\n"RESET);
    miaPosizione[1]=i;
    miaPosizione[2]=j;
    sendSocket(sock,miaPosizione,3);

```

```
sleep(1); inGioco=0; giocato=0;
break;
```

```
//i) info utenti
```

```
case 'i':
```

```
    for(k=0;k<5000;k++){
        lista[k]='\0';
    }
    sendSocket(sock,"i",1);
    if(read(sock,lista,5000)<0){
        error("error read\n");
    }
    stampa(lista);
    pauseT();
```

```
    for(k=0;k<5000;k++){
        lista[k]='\0';
    }
```

```
    sendSocket(sock,"k",1);
    if(read(sock,lista,5000)<0){
        error("errore read lista coordinate\n");
    }
    stampaMappa(Mappa,i,j,id_corrente,lista);
    break;
```

```
//t) info tempo
```

```
case 't':
```

```
    sendSocket(sock,"t",1);
    if(read(sock,&tempo,sizeof(int))<0){
        error("error read\n");
    }
```

```
    int tempoRimanente=tempoTot-tempo;
```

```
    sprintf(tempo_loc, GREEN "Tempo Trascorso :%d(sec)" RESET RED "\n\nTempo  
Rimanente :%d(sec)\n" RESET, tempo, tempoRimanente);
```

```
    stampa(tempo_loc);
    pauseT();
```

```
    for(k=0;k<5000;k++){
        lista[k]='\0';
    }
```

```
    sendSocket(sock,"k",1);
    if(read(sock,lista,5000)<0){
```

```

        error("errore read lista coordinate\n");
    }
    stampaMappa(Mappa,i,j,id_corrente,lista);
    break;

    //caso default
    default :
        stampa(RED"Inserimento Non Valido\n"RESET); pulisciBuffer(); break;

    }
}while(inGioco);

sleep(1);
pauseT();
}

```

```

// _____
_____ FUNZIONE

```

```

void stampaMappa(char Mappa[][20],int pos1, int pos2,int id_corrente,char *otherplayers){

    int i=0, j=0;
    char messaggio[256];
    char flag[256];

    //pausaGioco();
    for(i=0;i<20;i++){
        for(j=0; j<20; j++){
            if(i==pos1 && j==pos2);
            else
                Mappa[i][j]='-';
        }
    }

    //analisi stringa nemici
    int in;
    int temp=0;
    int nem[40];
    int indNem=0;

    if(otherplayers != NULL){

        //scorro la stringa
        for(in=0 ; in<strlen(otherplayers); in++){

```

```

//se e un numero
if(otherplayers[in]!='?' && otherplayers[in]!='&' && otherplayers[in]!='\0' ){
    temp = 0; //svuoto temp
    temp = (int)otherplayers[in]-'0'; // lo salvo

    // solo se dopo c'è un altro numero faccio questo
    if(otherplayers[in+1]!='?' && otherplayers[in+1]!='&' && otherplayers[in+1]!
    ='\0'){
        temp = 0; //svuoto temp
        temp = (int)otherplayers[in+1]-'0'; // lo salvo
        temp = temp+10; // lo incremento di 10
        in++; //in questo caso avanzo di uno (me ne fotto del primo)
    }
    //a questo punto in temp ho il numero giusto (1 cifra o 2 cifre)
    //dopo [in] c'è sicuro un carattere

    if(otherplayers[in+1]=='?' || otherplayers[in+1]=='&' || otherplayers[in+1]=='\0'){
        nem[indNem]=temp;
        indNem++;
    }
}
} //fine for

```

```

int indNem2; //indice array nemico (per scorrere)
int mappaX;
int mappaY;

```

//a questo punto ho un array di interi dove ogni 2 caselle sono [x][y][x][y]...

```

for (indNem2=0;indNem2<indNem;indNem2++){
    mappaX= nem[indNem2];
    indNem2++;
    mappaY= nem[indNem2];
    //segno il nemico in posizione giusta
    Mappa[mappaX][mappaY] = 'U';
}
} //fine if !=NULL
//fine posizionamento nemici

```

```

write(0,"\n",sizeof("\n"));
write(0,"\t  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 \n",sizeof("\t  0 1
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 \n"));

```

```

for(i=0; i<20; i++){
    if(i<10){

```

```

        sprintf(flag, "\t %d[ ", i);
    } else {
        sprintf(flag, "\t %d[ ", i);
    }
    stampa(flag);

    for(j=0; j<20; j++){
        if(Mappa[i][j]=='U' && (i==pos1) && (j==pos2)){
            stampa(BLU"@RESET");
        }
        else if(Mappa[i][j]=='U' && (i!=pos1) && (j!=pos2)){
            stampa("@");
        }
        else {
            write(0, &Mappa[i][j], 1);
        }
        if(j==19){
            write(0, " ", sizeof(" "));
        } else {
            write(0, " ", sizeof(" "));
        }
    }
    if(i==13){
        write(0, "]\n", sizeof("]"));
    } else if(i==14){
        write(0, "]\n", sizeof("]"));
    } else if(i==15){
        write(0, "]\n", sizeof("]"));
    } else if(i==16){
        write(0, "]\n", sizeof("]"));
    } else if(i==17){
        write(0, "]\n", sizeof("]"));
    } else if(i==18){
        write(0, "]\n", sizeof("]"));
    } else if(i==19){
        write(0, "]\n", sizeof("]"));
    } else {
        write(0, "]\n", sizeof("]\n"));
    }
}
sprintf(messaggio, YELLOW"Posizione [%d][%d] ID partita:
[%d])\n"RESET, pos1, pos2, id_corrente);
printf("w) Sopra a) Sinistra s) Sotto d) Destra t) Tempo i) Info c) Logout \n");
stampa(messaggio);
} //fine funzione stampaMappa

```

```
//
```

```
_____FUNZIONE
```

```
void sendSocket(int conn,char *mess,int nbytes)
{
    if(write(conn,mess,nbytes)<0){
        error("Errore write\n");
    }
}
```

```
//
```

```
_____FUNZIONE
```

```
void stampaVittoria(){
    int i=0;
    system("clear");
    signal(SIGINT,sighandler);
    signal(SIGQUIT,sighandler);
    signal(SIGHUP,sighandler3);
    signal(SIGSTOP,sighandler3);
    signal(SIGTERM,sighandler3);
    signal(SIGPIPE,sighandler4);
    signal(SIGTSTP,sighandler3);
    stampa(GREEN"!!!!!!!!!!!!!!!!!!!!\n");
    stampa("* ASSURDO HAI VINTO !!! *\n");
    stampa("!!!!!!!!!!!!!!!!!!!!\n"RESET);
    sleep(1);
    system("clear");
    sleep(1);
    i++;
}
```

```
//
```

```
_____FUNZIONE
```

```
void stampaSconfitta(){
    int i=0;
    system("clear");
    signal(SIGINT,sighandler);
    signal(SIGQUIT,sighandler);
    signal(SIGHUP,sighandler3);
    signal(SIGSTOP,sighandler3);
    signal(SIGTERM,sighandler3);
```

```
signal(SIGPIPE,sighandler4);
signal(SIGTSTP,sighandler3);
```

```
stampa(GREEN"*****\n");
stampa("* >>>> BOMBA ATOMICA <<<< *\n");
stampa("*****\n"RESET);
sleep(1);
system("clear");
sleep(1);
i++;
```

```
}
```

```
// _____
_____ FUNZIONE
```

```
void pauseT(){
    stampa("\nPremere Invio Per Continuare...\n");
    pulisciBuffer();
}
```